

Workshop on
HPC and Super-computing for Future Science Applications
June 4-6, 2013 • Brookhaven National Laboratory

HPC and HTC Studies in CMS

Peter Elmer
Princeton University

Overview

- This is a "grab-bag" talk of various things I think are interesting which fit under the very general rubric of "HPC and HTC" studies (which arguably covers many things)
- Some of this is recycled from recent ACAT13 presentations and/or presentations of CMS people
- Most of it is work by CMS, although a few things are "CMS + others"
- Much more than 30" of material, will skip some of it, but it is there for reference

Opportunistic Computing - General Considerations

- We are all interested in using opportunistically resources belonging to other people. Challenges include:
 - access to software
 - access to data (with or without local storage) and stageout
 - access to conditions
 - workload management issues
 - New processors (PowerPC) or limited supercomputer OS

Parked Data from 2012 and SDSC

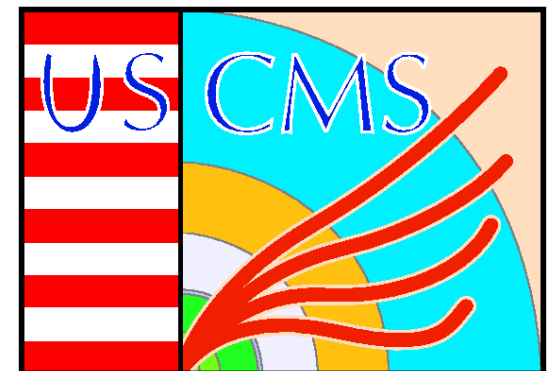
- During 2012 CMS wrote to tape a larger trigger rate than it could immediately process, with the intention of reconstructing it during the shutdown - "parked" data
- We got an allocation for a few weeks on Gordon at SDSC (NSF XSEDE) and decided to use it to process a couple of parked datasets of interest
 - <http://www.sdsc.edu/supercomputing/gordon/>
 - Compute: ~16K cores, 4GB/core
 - Storage: I/O nodes with SSD, 4PB parallel file system
 - InfiniBand
- Provision XSEDE resources as part of OSG ecosystem, Tier1 on-the-fly!

SDSC Gordon

- CMSSW software installed by hand on I/O node, served via NFS to compute nodes
- Frontier-squid with disk cache for conditions also installed on I/O node
- Storage accessible via gridFTP, no SRM. We used ~300TB for data on their Lustre filesystem (10Gbps peak transfer rate into SDSC).
- Data transferred in via PhEDEx, local stageout at SDSC with subsequent transfer via PhEDEx to FNAL.

SDSC Gordon - GlideinWMS setup

- No "grid interface" on Gordon, use ssh and OSG's BOSCO job submission manager (basically ssh-aware Condor-G)
- Dedicated factory - some hacking to support BOSCO and multicore jobs (got 16 core "whole node" job slot and fed it 16 jobs)
- Up to 4k jobs running at a time, with nearly 100% success rate
- Total 2 weeks to set up, processing took ~1 month, parked datasets processed and available for physics.

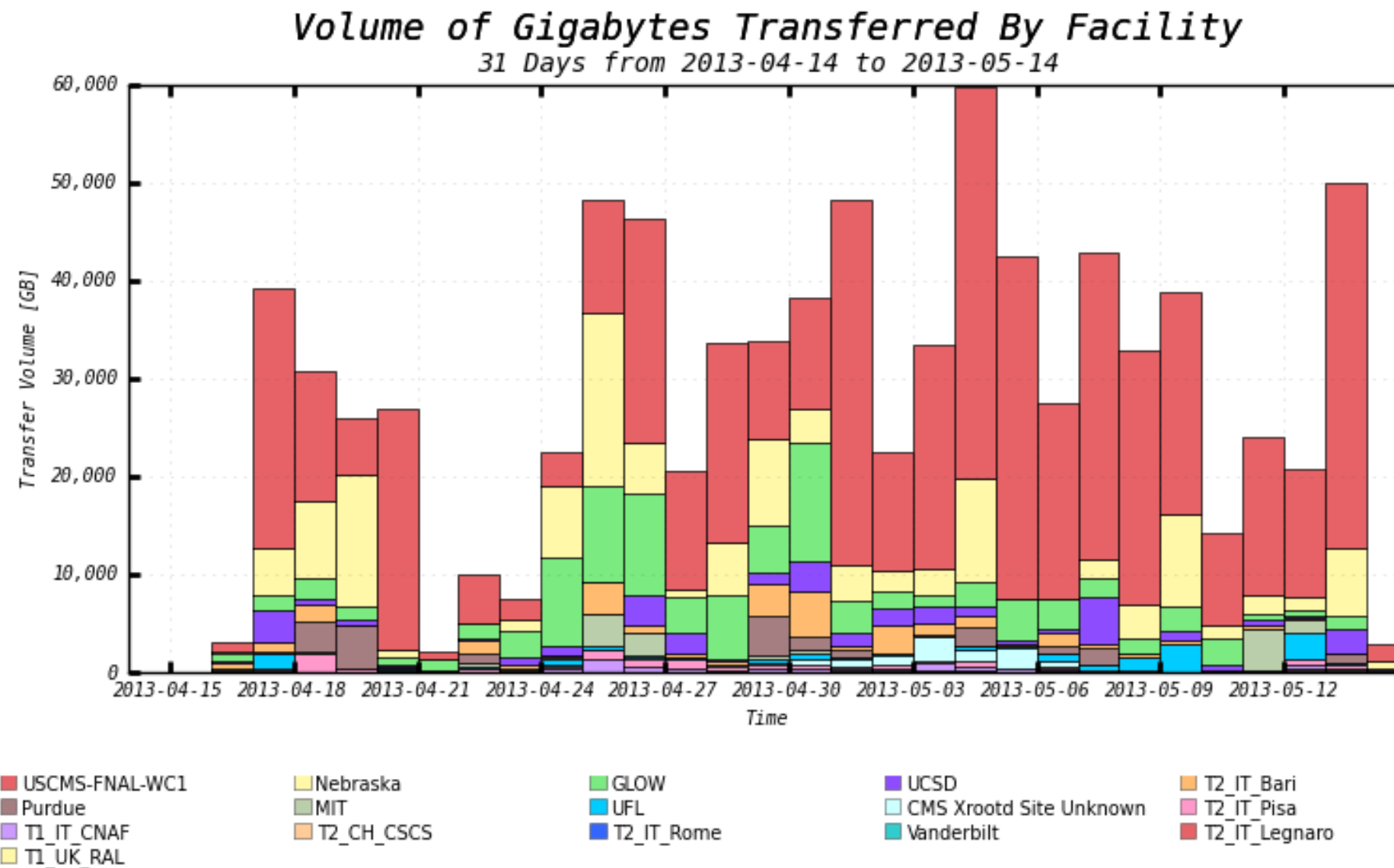


Next steps for opportunistic resource use

(First we take Manhattan,)

- SDSC was a nice first step that also accomplished a significant amount of useful and timely processing for CMS.
- Ultimately the goal is to have an even lighter footprint at the opportunistic site
- In addition to GlideinWMS (and BOSCO) two other relevant technologies: xrootd for data access and CVMFS/Parrot for software access
- Current work ongoing on opportunistic use of OSG resources: add use of Parrot wrapped jobs. Now testing with FermiGrid, another OSG site and a Russian site. Stageout to a CMS site.

Also using xrootd (AAA) for remote data access



Maximum: 59,874 GB, Minimum: 0.00 GB, Average: 27,406 GB, Current: 2,953 GB

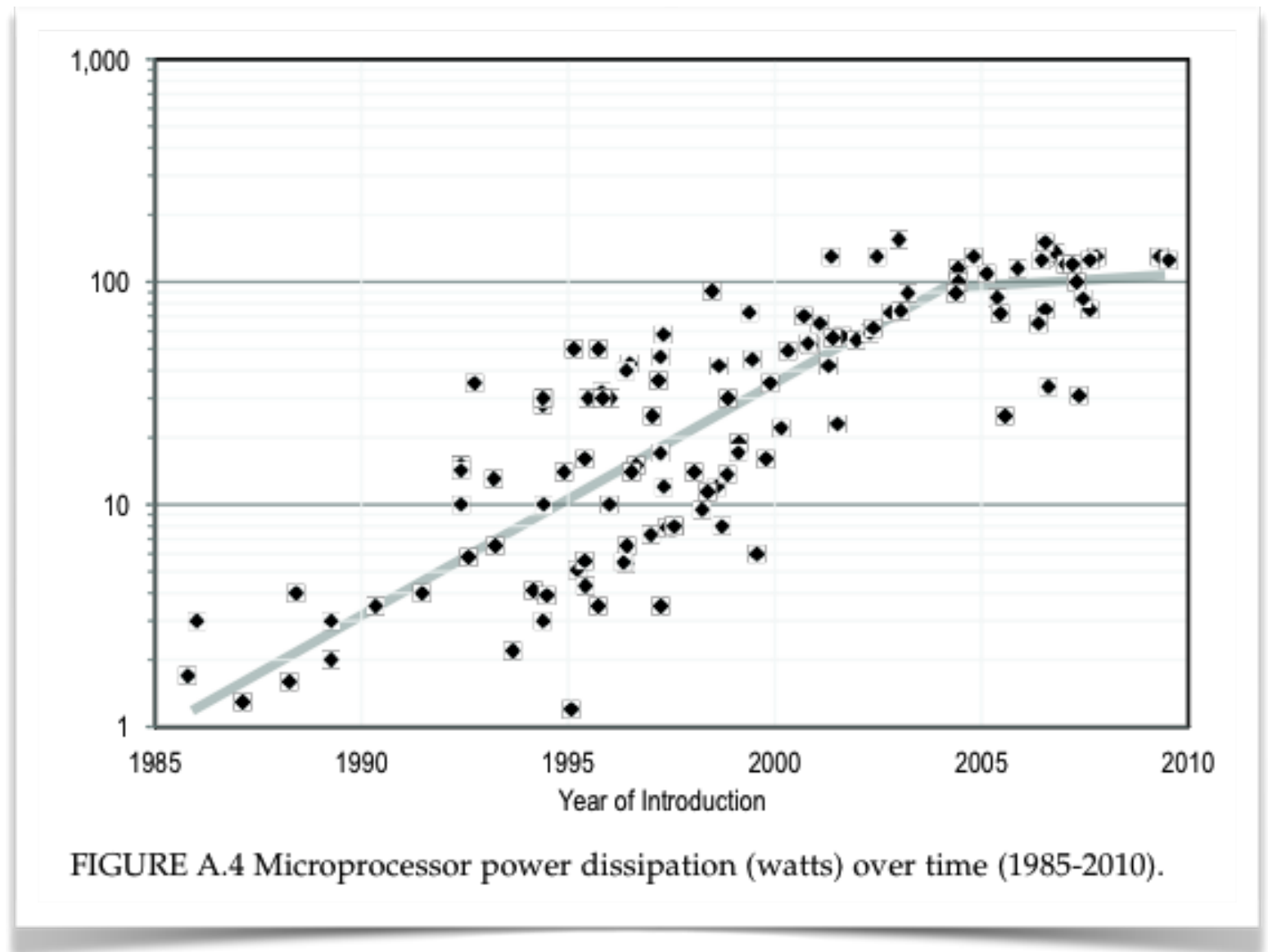
Dominated by a few sites at the moment. This is the beginning of the global data federation (standard sites, not opportunistic case), but demonstrates the technology.

NERSC

- Next step will be to bring all the pieces together to run on Carver/Hopper (eventually Edison?) at NERSC
- GlideinWMS, BOSCO, xrootd, CVMPS/Parrot, etc.
- For Hopper/Edison, we will need to use the "Cluster Compatibility Mode" instead of limited CLE/CNL environment
- <http://www.nersc.gov/users/computational-systems/hopper/cluster-compatibility-mode/>
- Work will begin with new person next month

Low Power Computing

- Over the past ten years processors have hit power limitations which place significant constraints on "Moore's Law" scaling.
- The first casualty was scaling for single sequential applications, giving birth to multi-core processors.

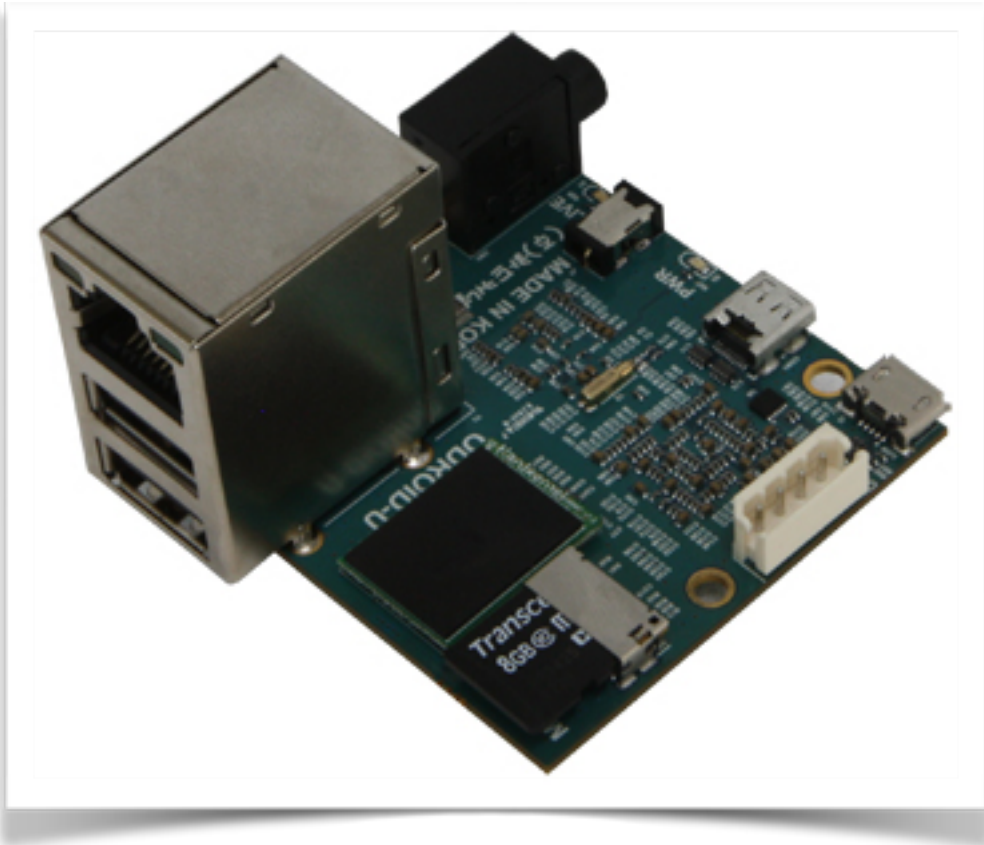


From: "The Future of Computing Performance:
Game Over or Next Level?"

ARM Servers

- This has led to the introduction of ARM-based servers in recent years, such as the Boston Viridis:
 - 192 cores in a 2U rack mount, consuming <300W
 - 48 quad-core nodes (1.4GHz Cortex-A9)
 - \$20k (reported)
- servers with the new ARMv8/64bit cores, expected next year, will likely be the product that will either create (or not) sufficient market share
- Dell "Copper" - 48 x quad-core=192 ARMv8 cores, 2GB/core, 750W in a 3U rack mount?

ARM Demonstrator - ODROID U2

- Initial tests done with a small 32bit/ ARMv7 development board
 - Exynos4412 Prime CPU
 - 1.7GHz Cortex-A9 quad core
 - 2GB L-DDR memory (total)
 - eMMC, microSD, 2xUSB2.0, 10/100Mbps Ethernet
 - \$89 (~\$233 with cables, cooling fan, 64GB eMMC, power adaptor, ...)
- 
- Fedora 18 ARMv7-A, hard floats, gcc 4.8, ODROID kernel

Building for ARM

- Early build attempts done with QEMU. Slow and buggy.
- Now we have a test board: cross compilation or native builds?
- If we eventually do have proper ARMv8/64bit servers with sufficient throughput for application use, we should be able to build natively.
- CMS has also invested over the years in optimizing its build system at many levels.
- The ODROID-U2 is actually reasonably powerful, so try a native build!

Build Issues

- No Oracle. But by construction no standard CMS grid-capable workflows can depend on Oracle. Affects a few special things.
- Minor compilation configuration issues: -m32/-m64 don't work, x86-ish assumptions leading to attempts to use SSE/AVX
- Signedness problems for char/bit-fields (Intel signed, ARM unsigned)
- Compilation of some translation units exhausted virtual memory (mostly ROOT dictionaries: refactor...)
- Patch needed for ROOT Cintex trampoline, plus one patch for dictionaries and some runtime issues being investigated

Build Status

- All externals build except Oracle and one online-only package
- 99% of CMSSW builds: a few remaining packages require Oracle plus a few being iterative broken/fixed as we sort out various last issues.
- All build recipes/patches available from:
 - `git://github.com/cms-sw/cmsdist.git`
 - branch "IB/CMSSW_6_2_X/fc18_armv7hl_gcc480"

Build Times on ODROID-U2

- ~4 hours mostly for gcc 4.8.0, but also a small set of basic things we need for packaging:
 - rpm, apt, zlib, ncurses, nspr, sqlite, etc.
- ~12 hours for all other "externals":
 - ROOT, Geant4, Python, Fastjet, Valgrind, gdb, boost, Qt, all generators, etc. Total of ~125 packages.
- ~25.5 hours for CMS software (CMSSW) - 3.5MSLOC of C++, plus generated ROOT dictionaries

First Benchmarks - Simulation (no output)

Type	Cores	TDP Power	Events/ min/core	Events/ min/Watt
Exynos4412 Prime @ 1.704GHz	4	4W?	1.14	1.14
Xeon L5520 @ 2.27GHz	2x4	120W?	3.50	0.23
Xeon E5-2630L @ 2.0GHz	2x6	190W?	3.33	0.21

First Benchmarks - Simulation (no output)

Type	Cores	full Power	Events/ min/core	Events/ min/Watt
Exynos4412 Prime @ 1.704GHz	4	6W?	1.14	0.76
Xeon L5520 @ 2.27GHz	2x4	240W	3.50	0.12
Xeon E5-2630L @ 2.0GHz	2x6	270W	3.33	0.15

Benchmarks - Notes

- These are *very* quick and dirty benchmarks, this is a work in progress. Numbers are "indicative", not final. Only very basic checks have been that results are consistent. ROOT output is still off.
- For power I used the TDP numbers from www.cpubenchmark.net, plus the quoted number for the ODROID (roughly measured by us), obviously ***not*** the total power cost especially for the Xeon servers. For those (second table "full power") I used some numbers from Bernd.
- I used one Nehalem (Q1 2010 release) and one Sandy Bridge (Q2 2012) "L" machine, both at CERN, vocms101 and vocms18. HT was on for the latter, but I have done just quick single core benchmark tests.

Porting IgProf to ARM?

- IgProf (igprof.org) is a sampling performance and memory profiler. Some notes on an ARMv7 port:
- ARM assembly much simpler than the `x86_64` one, all instructions are 32bit long: easier to decode. Documentation is excellent.
- However its RISC-ness introduces a few new quirks to be treated when instrumenting (conditional execution, linker peculiarity, less space for the actual instrumentation in the preamble).
- RDTSC instruction equivalent is not available in user mode.
- `libunwind` works out of the box, performance to verify

Multithreaded Framework

- We are currently evolving the CMS event processing framework to allow for multithreaded execution and in particular:
 - parallel execution of multiple events
 - parallel execution of modules (algorithms: producers, analyzers)
 - parallel execution of code within a module
- Underlying technology choice is TBB, eventual use TBB-enabled multithreaded externals (Geant4, etc.)

Multithreaded Framework

- Objectives:
 - Reduce overall memory requirements
 - Reduce number of open files, connections, jobs
 - Reduce number of output files
 - (Eventually) possibly improve throughput through better behavior of fine-grained parallelism on memory hierarchy
- Overall a more scalable application for the long run for x86-64, eventually ARM or low power Intel, Xeon Phi?

Multithreaded Framework

- By default code is not run in parallel, requires modification
- Using Clang/LLVM static analyzer to look for thread unsafe constructs
- Framework ready by fall, expect to deploy for production in early 2014

Numerical Computing

- A confluence of things has led to a renewed interest in HEP in the numerical aspects of computing: the transition to x86-64, "bazaar" evolution of gcc4, various effects of microprocessor power limitations
- CERN OpenLab/Intel/PH-SFT workshops on numerical computing, e.g.
 - <http://indico.cern.ch/conferenceTimeTable.py?confId=247985#20130527>
- Interest in floating point vectorization, required accuracy, math libraries, etc.

SLC5 math library - "bug fix"

27-8-2012: <http://rhn.redhat.com/errata/RHSA-2012-1207.html>

* Previously, logic errors in various mathematical functions, including exp, exp2, expf, exp2f, pow, sin, tan, and rint, caused inconsistent results when the functions were used with the non-default rounding mode. This could also cause applications to crash in some cases. With this update, the functions now give correct results across the four different rounding modes. (BZ#839411)

Excellent:

Now results are consistent!

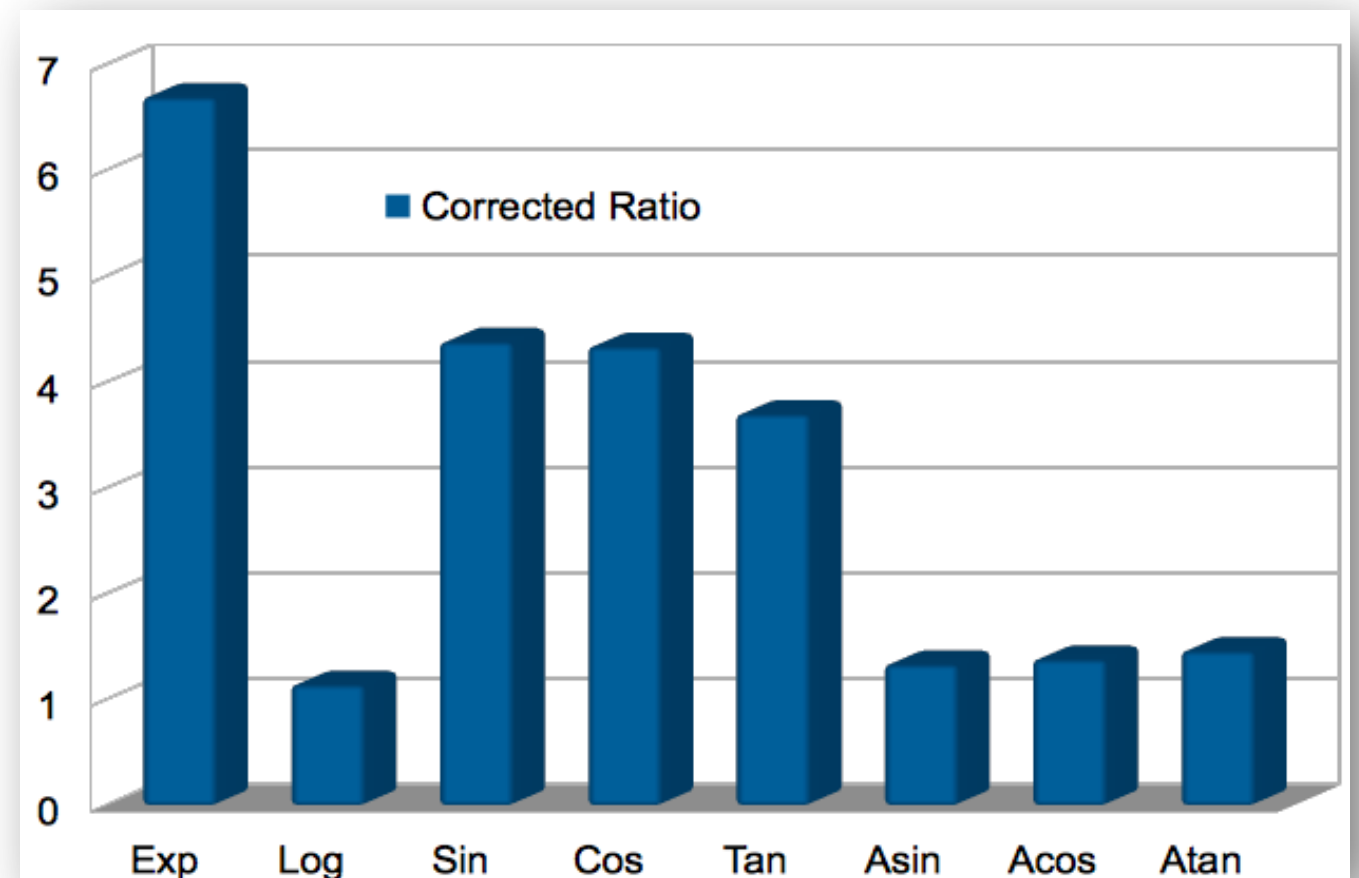
(**feraiseexcept** function used to raise fp exception when result “unprecise”)

From D. Piparo (V. Innocente, T. Hauth)

Effect of libm change

Wait: how much tax payers' money does this cost?

The modified routines cause a slow-down of a factor >6 for Exp and important ones for Sin, Cos and Tan.



Nice to have such a solid reference, but can we afford that in our production software?

Probably not... [What are the alternatives?](#)

From D. Piparo (V. Innocente, T. Hauth)

A Selection of Alternatives

A plethora of different products are available, for example:

- Intel's SVML, IMF, MKL (commercial)
- AMD Libm (free)
- VDT (Vectorised maTh: free and open source)



Differences in the implementations but common underlying principle:

Trade off between accuracy and speed of execution

From D. Piparo (V. Innocente, T. Hauth)

- An **open source** math library library, LGPL3 licence
- Single and Double precision of (a)sin, (a)cos, sincos, (a)tan, atan(2), log, exp and 1/sqrt
- **Fast, approximate, inline** (see following slide for the details)
- Symbols names are different from traditional ones: `vdt::fast_<name>`
 - Do not force drop-in replacement!
- **Autovectorisable** since gcc 4.7
 - **Array signatures** available: calculate on multiple elements conveniently
 - Can be inserted in **autovectorised loops** (inline!)
- Inspired by the good old Cephys (and Quake III videogame)
- **Standard C code only** is used (no intrinsics): **portability guaranteed**
 - ARM, x86, GPGPUs, Xeon Phi, <future microarchitecture>



<https://svnweb.cern.ch/trac/vdt>



From D. Piparo (V. Innocente, T. Hauth)

Underlying principle behind VDT (and Cephes): Padé' Approximants

The "best" **approximation of a function by a rational function** of a given order

- Often better approximation than a truncated Taylor series

Padé approxi

$$R(x) = \frac{\sum_{j=0}^m a_j x^j}{1 + \sum_{k=1}^n b_k x^k} = \frac{a_0 + a_1 x + a_2 x^2 + \dots + a_m x^m}{1 + b_1 x + b_2 x^2 + \dots + b_n x^n}$$

which agrees to the highest

$$\begin{aligned} f(0) &= R(0) \\ f'(0) &= R'(0) \\ f''(0) &= R''(0) \\ &\vdots \\ f^{(m+n)}(0) &= R^{(m+n)}(0) \end{aligned}$$



From D. Piparo (V. Innocente, T. Hauth)

Speed: VDT Vs Libm

Double
Precision

Testbed:

SLC6-GCC47, Core i7-3930K CPU @

3.20GHz

Speed

*Time in ns per
value calculated*

Function	Libm	VDT	VDT SSE	VDT AVX
Exp	16.7	6.1	3.8	2.9
Log	34.9	12.5	5.7	4.2
Sin	33.7	16.2	6.0	5.7
Cos	34.4	13.4	5.4	5.1
Tan	46.6	12.5	6.3	5.6
Asin	23.0	10.3	8.6	8.1
Acos	23.7	11.0	8.2	8.1
Atan	19.7	11.0	8.3	8.3
Isqrt	9.3	6.7	3.0	2.1

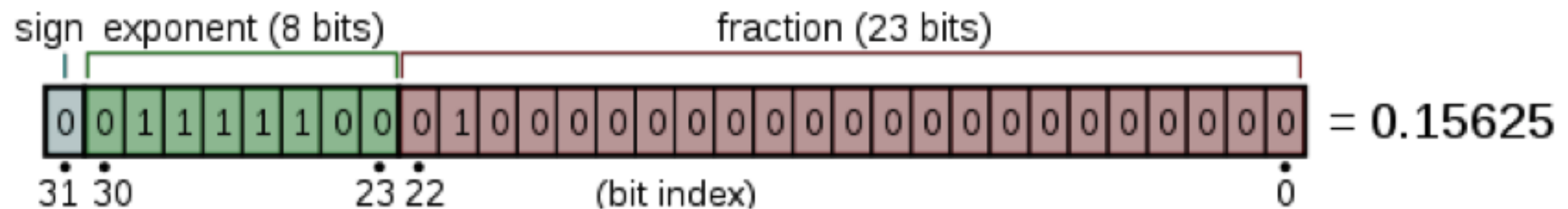
- Operative input range: [-5000, 5000]
- VDT scalar functions:
 - Speedups of ~4x achievable
- Speedup scalar \boxtimes SSE more significant than SSE \boxtimes AVX
 - Some overhead is present

Time in ns per value calculated

From D. Piparo (V. Innocente, T. Hauth)

Some Words about Accuracy

- Accuracy was measured comparing the results of Libm and VDT bit by bit with the same input
- Differences quoted in terms of most significant different bit
- In the end they are just 32 (64) bits which are properly interpreted!



A single precision floating point number

From D. Piparo (V. Innocente, T. Hauth)

Double
Precision

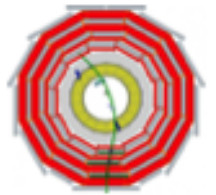
	MAX VDT	AVG VDT
Acos	8	0.39
Asin	2	0.32
Atan	1	0.33
Cos	2	0.25
Exp	2	0.14
Isqrt	2	0.45
Log	2	0.42
Sin	2	0.25
Tan	2	0.35

Approximate results, but ok for a wide range of applications

From D. Piparo (V. Innocente, T. Hauth)

Full Sim: Costs of the Functions 1/2

$H, A \rightarrow \tau\tau \rightarrow \text{two } \tau \text{ jets} + X, 60 \text{ fb}^{-1}$



Function	Runtime %	
<i>_ieee_754_log</i>	3.77	13.30
<i>_ieee_754_exp</i>	1.80	5.85
<i>_ieee_754_atan2</i>	1.74	0.75
<i>sincos</i>	0.60	0.37
<i>_ieee_754_pow</i>	0.51	0.45
<i>__exp1</i>	0.29	0.26
<i>_ieee_754_log10</i>	0.16	0.08
<i>_ieee_754_atan2f</i>	0.15	0.03
TOTAL	9.02	21.9

Performance profile of 2 jobs obtained:

- 1) 50 events (~5k seconds)
- 2) 1 event (310 seconds): estimator of the initialisation overhead

Numbers for 1) are in black in the table, the ones for 2) red in the table

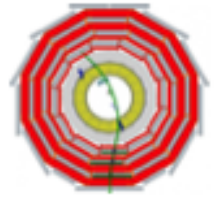
Self costs shown, callees are not considered!

T. Hawth

From D. Piparo (V. Innocente, T. Hawth)

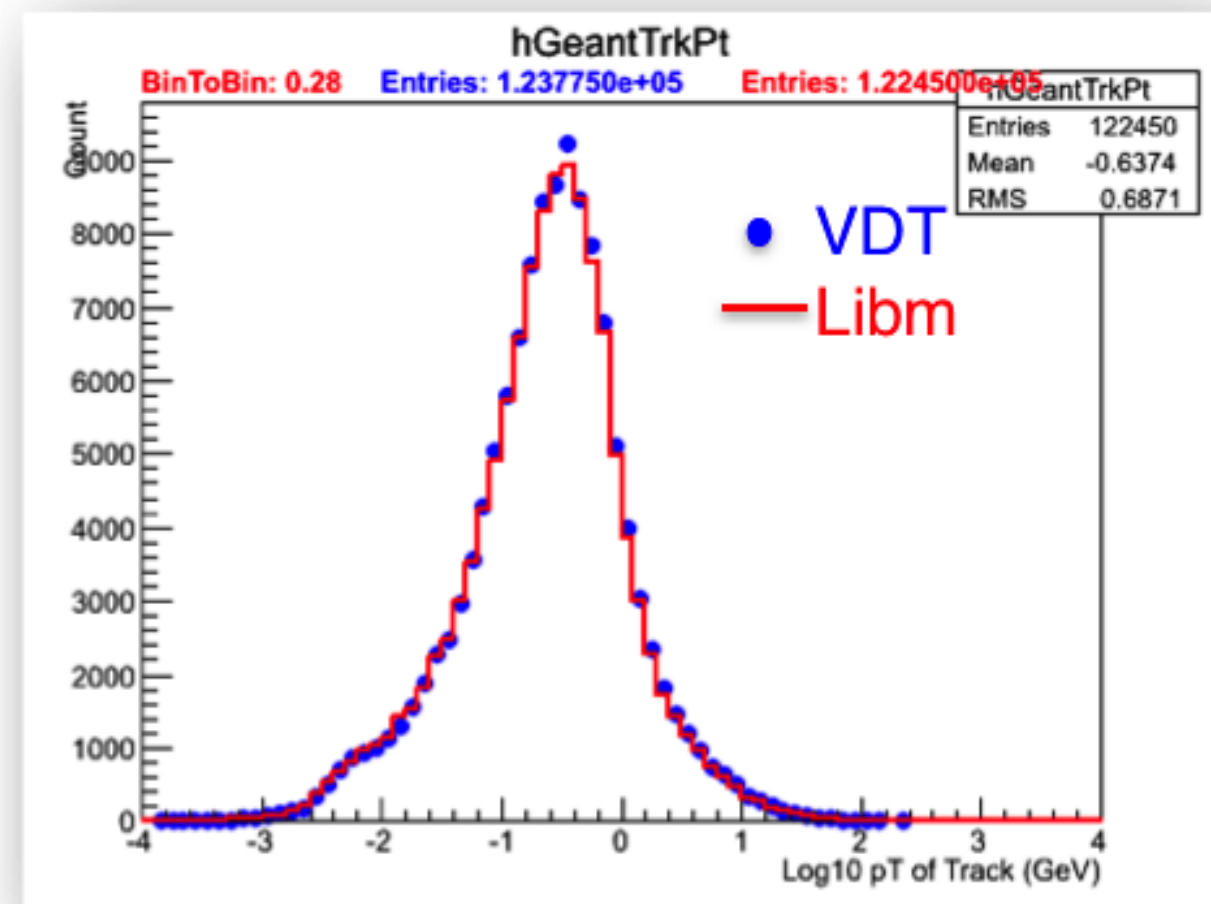
Result:

- 9% speedup achieved (FullSim 50 Events)
- 25% speedup achieved (FullSim1 Event – Initialisation cost)



Validation:

- Good compatibility of results assessed
 - Use standard CMS histograms
- Changes expected and found
 - Different accuracy of the functions!
 - Experts' validation must sign-off!



T. Hauth

From D. Piparo (V. Innocente, T. Hauth)

Checkpoint-Restart

- It is desirable in certain circumstances to "checkpoint" the state of a unix process, or set of processes, to disk with the possibility of restarting it at a later time.
- This can be done in an application-specific custom fashion, but it requires the addition and maintenance of dedicated code.
- A generalized technology capable of checkpointing all types of applications is thus desirable. In fact such technologies have been in use in High Performance Computing (HPC) and batch systems since more than 20 years.

Checkpoint-Restart - Interesting Use Cases

- Avoiding CPU-intensive initialization steps in frequently run applications, perhaps avoid need for conditions or other loading
- Reproducibility of problems in long running jobs for debugging
 - The application can be "replayed" from a point just before the error or crash, rather than from the beginning
- In situations where resources are being used opportunistically, it can be used to efficiently give access back to the "owner" and then later restart when resources are free again
- In interactive applications, the current state can be saved ("workspace")
- For long-running parallel applications sensitive to hardware failure, the state of calculations can be saved periodically to allow restart.

DMTCP

- Distributed MultiThreaded CheckPointing package (DMTCP), developed at Northeastern University (NEU), <http://dmtcp.sourceforge.net>

Key Features

Userspace checkpointing, no kernel-level access required

Checkpoints multithreaded applications

Checkpoints distributed applications

Can handle fork, exec, ssh, open file descriptors, TCP/IP sockets, etc.

Minimum runtime overhead

Optional compression of checkpoint images

Open source

Works on linux and supports a wide range of kernels

DMTCP - CMS Example

- Quick test with CMS Framework-based generation/simulation application, memory footprint ~750MB RSS
- ~10s required to create compressed checkpoint image, 220MB
- 1-2s for uncompressed checkpoint

```
Begin processing the 1st record. Run 1, Event 1, LumiSection 1 at 18-May-2013 05
:10:01.557 CEST
Begin processing the 2nd record. Run 1, Event 2, LumiSection 1 at 18-May-2013 05
:10:01.584 CEST
WARNING: G4QPDGToG4Particle is deprecated and will be removed in GEANT4 version
10.0.
Begin processing the 3rd record. Run 1, Event 3, LumiSection 1 at 18-May-2013 05
:10:10.014 CEST
Begin processing the 4th record. Run 1, Event 4, LumiSection 1 at 18-May-2013 05
:10:14.434 CEST
Begin processing the 5th record. Run 1, Event 5, LumiSection 1 at 18-May-2013 05
:10:18.362 CEST
```

Trigger checkpoint externally while processing event #5

DMTCP - CMS Example

- And restart:

```
vocms101> ./dmtcp_restart_script_a9bf217912ea647-48000-5196f087.sh
dmtcp_restart (DMTCP + MTCP) 2.0
Copyright (C) 2006-2011 Jason Ansel, Michael Rieker, Kapil Arya, and
Gene Cooperman
```

```
This program comes with ABSOLUTELY NO WARRANTY.
This is free software, and you are welcome to redistribute it
under certain conditions; see COPYING file for details.
(Use flag "-q" to hide this message.)
```

```
Begin processing the 6th record. Run 1, Event 6, LumiSection 1 at 18-May-2013 05
:15:53.486 CEST
```

```
Begin processing the 7th record. Run 1, Event 7, LumiSection 1 at 18-May-2013 05
:15:56.754 CEST
```

```
G4Fragment::CalculateExcitationEnergy(): WARNING
```

```
Fragment: A = 26, Z = 12, U = -3.520e-01 MeV IsStable= 1
```

```
P = (-9.200e+01,1.075e+02,-2.607e-01) MeV E = 2.420e+04 MeV
```



This was on x86-64, also ARM and Xeon Phi supported

The End

Enough already.....